Welcome to Course 2!

Today's talk:

Solving linear systems II

Relaxations & multigrid

Introduction - Model problems

Problem 1 Steady-state temperature distribution in 1-D
(for example in a stick)

$$\begin{cases} -u''+\sigma\, u = f, \quad 0<x<1, \sigma \geq 0 \\ u(0)=u(1)=0 \end{cases}$$

Question: How can we solve that problem?

$\Rightarrow$ Finite difference discretization

$$\begin{cases} \dfrac{-v_{j-1}+2v_j-v_{j+1}}{h^2}+\sigma\, v_j = f_j, & 1 \leq j \leq N-1 \\ v_0 = v_N = 0 \end{cases}$$

vectorial form of this problem:

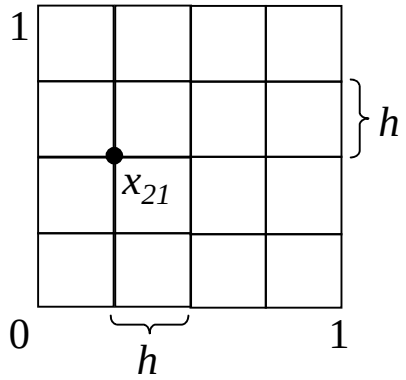
$$\underbrace{\frac{1}{h^2}\begin{bmatrix} 2+\sigma h^2 & -1 & & 0 \\ -1 & \ddots & & \\ & \ddots & & -1 \\ 0 & & -1 & 2+\sigma h^2 \end{bmatrix}}_{=: A \in \mathrm{IR}^{N-1 \times N-1}} \cdot \vec{v} = \vec{f}, \qquad \vec{v} \in \mathrm{IR}^{N-1}, \vec{f} \in \mathrm{IR}^{N-1}$$

→ A is tridiagonal, symmetric positive definite (s.p.d.)

Problem 2 Poisson's equation (2-D-case)

$$\begin{cases} -\Delta u(x,y) = -u_{xx} - u_{yy} = f(x,y) & \text{for } (x,y) \in \Omega := (0,1)^2 \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

Discretizing this equation by finite differences on an equidistant grid with meshsize $h > 0$ results in



$$\begin{cases} \dfrac{-v_{i-1,j} - v_{i,j-1} + 4v_{ij} - v_{i+1,j} - v_{i,j+1}}{h^2} = f_{ij} & , \ 1 \le i, j \le N-1 \\ v_{ij} = 0 & , \ i = 0 \vee j = 0 \vee i = N \vee j = N. \end{cases}$$

vectorial form of this problem:

$$\underbrace{\frac{1}{h^2}\begin{bmatrix} A & -E & & 0 \\ -E & \ddots & & \\ & \ddots & \ddots & -E \\ 0 & & -E & A \end{bmatrix}}_{\in \mathrm{IR}^{(N-1)^2 \times (N-1)^2}} \cdot \vec{v} = \vec{f}$$

with $\vec{v}, \vec{f} \in \mathrm{IR}^{(N-1)^2}, E \in \mathrm{IR}^{N-1 \times N-1}$ identity matrix,

$$A := \begin{bmatrix} 4 & -1 & & 0 \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{bmatrix} \in \mathrm{IR}^{N-1 \times N-1}$$

Conclusion

Both problems lead to big sparse linear equation systems

→ In special cases like Problem 1, there exist direct solvers with "good" costs (Problem 1: $O(N^2 log N)$)

→ <u>But:</u> Normally, direct solving is much too expensive for more complicated matrices

$\Rightarrow$ this motivates the idea of

<u>iterative/ relaxation solvers for linear equation systems</u>

Basic iterative solvers

Regard the linear equation system

(+) $Au = f,$ $A \in \mathrm{IR}^{n\times n}$ regular $, u \in \mathrm{IR}^n, f \in \mathrm{IR}^n$

Writing A as $A = D\text{-}L\text{-}U$, (where D is the diagonal, $-L$ the lower left and $-U$ the upper right part of A), it follows

$$Au = f \quad \Leftrightarrow \quad u = D^{-1}(L+U)u + D^{-1}f.$$

Jacobi-method

Let $v^{(0)} \in \mathrm{IR}^n$ be an initial guess as solution for (+). The iteration

$$v^{(i+1)} = P_j v^{(i)} + D^{-1}f \qquad \text{with } P_j := D^{-1}(L+U) \in \mathrm{IR}^{n\times n}$$

is called *Jacobi-method*; P_j is the so-called *iteration-matrix* of this method.

Weighted/ Damped Jacobi-method

Let $v^{(0)} \in \mathbb{R}^n$ be an initial guess for (+) and let $\omega \in \mathbb{R}$ (normally $\omega \in (0,1)$). The following iteration scheme is called *weighted* or *damped Jacobi-method*:

$$v^* = P_j v^{(i)} + D^{-1} f$$
$$v^{(i+1)} = (1-\omega)\ v^{(i)} + \omega\ v^*$$

Remarks:

- ω is called *weighting factor*.
- The iteration-matrix of this method is $P_\omega := (1-\omega)E + \omega\ P_j$

$$\longrightarrow \quad v^{(i+1)} = P_\omega\ v^{(i)} + \omega\ D^{-1} f$$

Since now: $Au = f \quad \rightarrow \quad u = D^{-1}(L+U)u + D^{-1} f$

New idea: $Au = f \quad \rightarrow \quad u = (D-L)^{-1}Uu + (D-L)^{-1} f$

Gauss-Seidel-Relaxation

Let $v^{(0)} \in \mathbb{R}^n$ be an initial guess for (+). The following iteration scheme is called *Gauss-Seidel-Relaxation*:

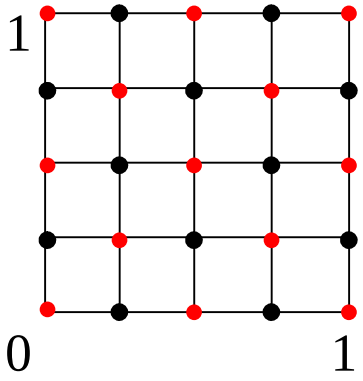
$$v^{(i+1)} = P_G \, v^{(i)} + (D-L)^{-1} f,$$

where $P_G := (D-L)^{-1}U$.

Red-black-Gauss-Seidel-Relaxation

- regard the grid of Problem 2
- split that grid into red and black grid points
- the Laplace stencil for one grid point just depends on its direct neighbours

  → do GS-relaxation on grid points v_{2j} first; after that do same update-method to v_{2j+1}



Difference between Gauss-Seidel- and Jacobi-iteration

Writing the iteration schemes for every single unknown $v_k^{(i+1)}$, we get:

Jacobi: $$v_k^{(i+1)} = \frac{1}{a_{kk}} \left(f_k - \sum_{\substack{j=1, \\ j \neq k}}^{n} a_{kj} v_j^{(i)} \right)$$

Gauss-Seidel: $$v_k^{(i+1)} = \frac{1}{a_{kk}} \left(f_k - \sum_{j>k} a_{kj} v_j^{(i)} - \sum_{j<k} a_{kj} v_j^{(i+1)} \right)$$

→ Jacobi-iteration invariant under changing the order of the unknowns
Gauss-Seidel-iteration changes by using a new ordering of the unknowns

Analysis for the basic solvers

So far, we found some possible iteration methods.

Questions:

- In which cases do they converge to the exact solution?
- How fast do they converge?
- What initial guess for $v^{(0)}$ is preferable?

...

Regard the general iteration scheme $v^{(i+1)} = Pv^{(i)} + g$; let u be the exact solution of $Au = f$ and let $e^{(i)} = u - v^{(i)}$ be the error of the iterated solution in step i to the exact solution u. It holds:

- u exact solution $\Leftrightarrow$ $u = Pu + g$ ($\rightarrow u$ fixpoint)

  $\rightarrow$ Idea: proving convergence by Banach's fixpoint theorem

- $e^{(n)} = Pe^{(n-1)} = \ldots = P^n e^{(0)} \quad \Rightarrow \quad \left\| e^{(n)} \right\| \leq \left\| P \right\|^n \cdot \left\| e^{(0)} \right\|$

So: $\left\| P \right\| < 1 \quad \Rightarrow \quad \left\| e^{(n)} \right\| \xrightarrow{n \to \infty} 0$

Furthermore, let $\rho(P) := \max_{\substack{\lambda \text{ eigenvalue} \\ \text{of } P}} |\lambda(P)|$ be the spectral radius of P.
Then, it holds:

$$\lim_{n\to\infty} P^n = 0 \quad \Leftrightarrow \quad \rho(P) < 1$$

$\longrightarrow$ Iteration-method converges, if $\rho(P) < 1$.
$\rho(P)$ is called *convergence factor*.

How many iterations do we need to achieve $\left\| e^{(n)} \right\| \leq 10^{-d}$ for given $d \in \mathbb{N}$?

$$n \geq -\frac{d}{\log_{10} |\rho(P)|}$$

Fourier analysis of damped Jacobi-iteration in case of (modified) Problem 1

$$\longrightarrow \quad P_\omega = E - \frac{\omega}{2}\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}$$

$$\longrightarrow \quad \lambda_k(P_\omega) = 1 - 2\omega \sin^2\left(\frac{k\pi}{2N}\right) \quad \text{eigenvalues of } P_\omega,$$

$$w_{k,j} = \sin\left(\frac{jk\pi}{N}\right) \quad \text{corresponding eigenvectors for } 1 \le k \le N-1, \quad 0 \le j \le N$$

Let $e^{(0)}$ be initial error, choose $\omega \in (0,1)$ to guarantee convergence.

$\longrightarrow$ Write $e^{(0)}$ as linear combination of the eigenvectors

$$e^{(0)} = \sum_{k=1}^{N-1} c_k w_k, \quad c_k \in \mathbb{R}$$

$$\Rightarrow \quad e^{(n)} = P_\omega^n e^{(0)} = \sum_{k=1}^{N-1} c_k P_\omega^n w_k = \sum_{k=1}^{N-1} c_k \lambda_k^n w_k$$

Since $\lambda_k = 1 - 2\omega \sin^2\left(\frac{k\pi}{2N}\right)$, it holds:

- For small k ($\longrightarrow 1 \le k \le N/2$), λ_k is near to 1 $\Rightarrow$ $\lambda_k^n w_k$ remains big

  Since $w_{k,j} = \sin\left(\frac{jk\pi}{N}\right)$, these components consist of

  a small number of sine waves

  $\longrightarrow$ $w_{k,j}$ are called *low-frequency* or *smooth modes*

- For large k ($\longrightarrow N/2 \le k \le N-1$), λ_k is very small

  $\Rightarrow$ $\lambda_k^n w_k$ is very small

  Since $w_{k,j} = \sin\left(\frac{jk\pi}{N}\right)$, these components consist of

  a large number of sine waves

  $\longrightarrow$ $w_{k,j}$ are called *high-frequency* or *oscillatory modes*

Thus, it holds:

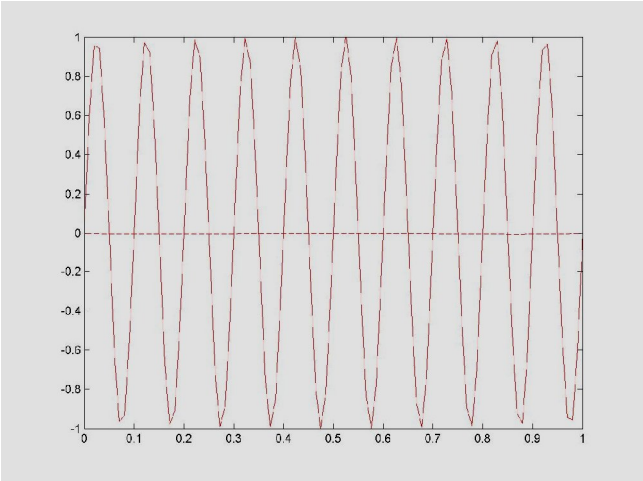
- High-frequency errors are damped very good
- Low-frequency errors are reduced very slowly

by iterative methods like Jacobi, Gauss-Seidel etc.

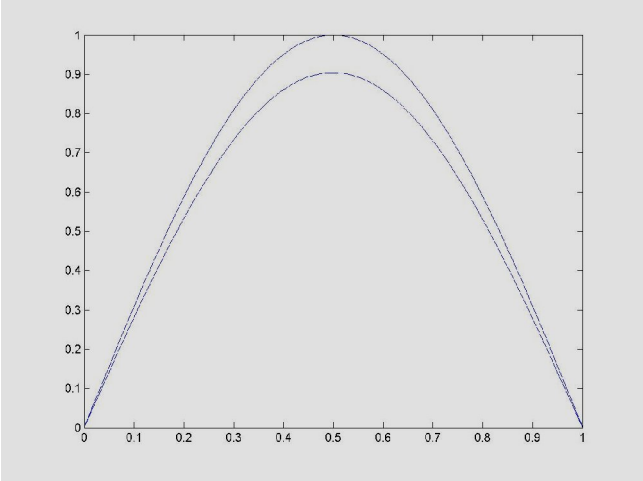
Example:

$$-f'' = 0 \quad \text{on (0,1)} \qquad f(0) = f(1) = 0$$

$\longrightarrow$ Discretize by FD, take $N = 100$, and do 100 iteration steps with Gauss-Seidel-relaxation



$$v^{(0)} = \sin\left(\frac{20\pi}{N-1}\right)$$



$$v^{(0)} = \sin\left(\frac{\pi}{N-1}\right)$$

Code example 1

Task:

Write a program *"itsolve.cpp"* that discretizes the PDE

$$-\Delta u = 32\pi^2 \cos(4\pi x)\cos(4\pi y) \quad \text{on } \Omega := (0,1)^2$$
$$u = \cos(4\pi x)\cos(4\pi y) \quad \text{on } \partial\Omega$$

by finite differences on $\overline{\Omega}$ and solves the discrete problem by different iterative methods.
The programm should finally create the file *"itsolution.txt"* and write there the approximate solution in order to visualize it later.

→ Let's have a look at the code!

Multigrid schemes

So far: Relaxation schemes which solve linear equation systems

Disadvantage: No damping of low-frequency components
⟶ Problem for bad initial guesses

New goal: Improve convergence behaviour/
improve initial guess

Possible strategies: 1) Nested iteration
2) Coarse grid correction

Strategie 1: Nested iteration

Main idea: Use a coarse grid to obtain a good initial guess for the “original” grid

Iteration scheme

Step 1 Generate very coarse grid
Set minimal mesh size $h > 0$
Set tolerance $TOL > 0$ for the error

Step 2 Relax $Au = f$ on current level

Step 3 If (current grid $\neq$ “finest grid”)
transfer u to a finer grid
go to step 2
else
relax $Au = f$, until it holds
$|| r || < TOL$ for the residual $r = f - Av$

Strategie 2: Coarse grid correction
(At first: 2-grid-method)

Main idea: Use the residual equation $r = f - Av$ to relax on the error by working on different grid levels

Iteration scheme

	Step 1	Do some relaxation steps for $A^h u^h = f^h$ on the fine grid Ω^h (*pre-smoothing*)
restriction		Compute $r^h = f^h - A^h v^h$
	Step 2	Restrict r^h to the coarser grid Ω^{2h}
	Step 3	Do relaxations with initial guess $e^{2h} = 0$ to solve the system $A^{2h} e^{2h} = r^{2h}$ on Ω^{2h}
prolongation/ interpolation	Step 4	Prolongate e^{2h} on the fine grid Ω^h
	Step 5	Correct the original solution $v^h := v^h + e^h$
		Do some relaxation steps on Ω^h (*post-smoothing*)

For following discussion:

Given: v^h

Assumptions: Only transfer actions from $\Omega^h \rightarrow \Omega^{2h} \rightarrow \Omega^h$

Question: How to transfer v^h to coarser/finer grid?

Restriction

Two possibilities: 1) Injection
2) Full weighting restriction

1) Injection

Take value of every second grid point

$$v_j^{2h} = v_{2j}^h$$

$\longrightarrow$ Most easiest way of restriction

But: bad interpretation of fine grid data on the coarse grid

2) Full weighting restriction

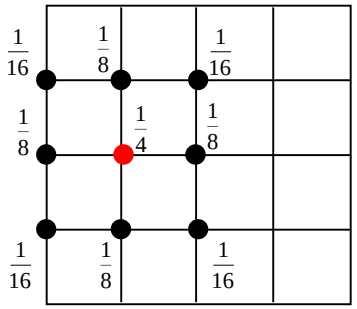
linear restriction operator $\begin{cases} I_h^{2h} : \Omega^h \to \Omega^{2h} \\ I_h^{2h} v^h = v^{2h} \end{cases}$ with better "properties" than restriction scheme 1)

1-D case:

$$v_j^{2h} = \frac{1}{4}\left(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h\right) \qquad 1 \le j \le N/2 - 1$$

2-D case:

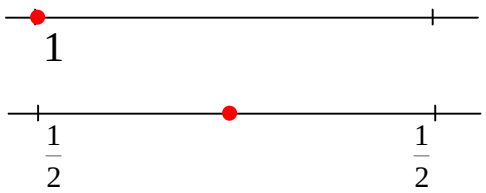
$$\begin{aligned} v_{ij}^{2h} = & \frac{1}{16}\left(v_{2i-1,2j-1}^h + v_{2i-1,2j+1}^h + v_{2i+1,2j-1}^h + v_{2i+1,2j+1}^h\right) \\ & + \frac{1}{8}\left(v_{2i,2j-1}^h + v_{2i,2j+1}^h + v_{2i-1,2j}^h + v_{2i+1,2j}^h\right) \\ & + \frac{1}{4} v_{2i,2j}^h \qquad 1 \le i, j \le N/2 - 1 \end{aligned}$$



Prolongation/ Interpolation (also by a linear operator $\begin{cases} I_{2h}^{h} : \Omega^{2h} \to \Omega^{h} \\ I_{2h}^{h} v^{2h} = v^{h} \end{cases}$)

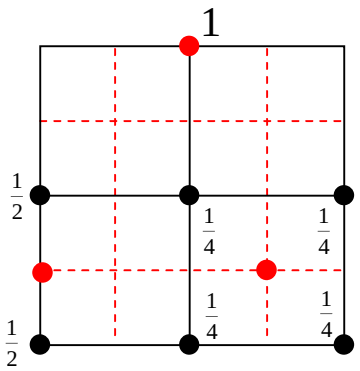
1-D case:

$$v_{2j}^{h} = v_{j}^{2h}$$
$$v_{2j+1}^{h} = \frac{1}{2}\left(v_{j}^{2h} + v_{j+1}^{2h}\right) \qquad 1 \le j \le N/2 - 1$$



2-D case:

$$v_{2i,2j}^{h} = v_{ij}^{2h}$$
$$v_{2i,2j+1}^{h} = \frac{1}{2}\left(v_{i,j}^{2h} + v_{i,j+1}^{2h}\right)$$
$$v_{2i+1,2j}^{h} = \frac{1}{2}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h}\right)$$
$$v_{2i+1,2j+1}^{h} = \frac{1}{4}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}\right) \qquad 1 \le i, j \le N/2 - 1$$



So far: 2-grid-algorithm

- Relax on $Au = f$ on Ω^h, calculate $r = f - Av$ → not as cheap, but unavoidable
- Restrict r^h on Ω^{2h} by I_h^{2h} → cheap costs
- Calculate e^{2h}, from $A^{2h} e^{2h} = r^{2h}$ → for higher levels *still expensive*
- Prolongate the error e^{2h} to the grid Ω^h by I_{2h}^h → cheap costs
- Correct original solution $v^h := v^h + e^h$ → cheap costs
- Relax again on $Au = f$ on Ω^h

Question: How to solve $A^{2h}e^{2h} = r^{2h}$ in a cheaper way ?

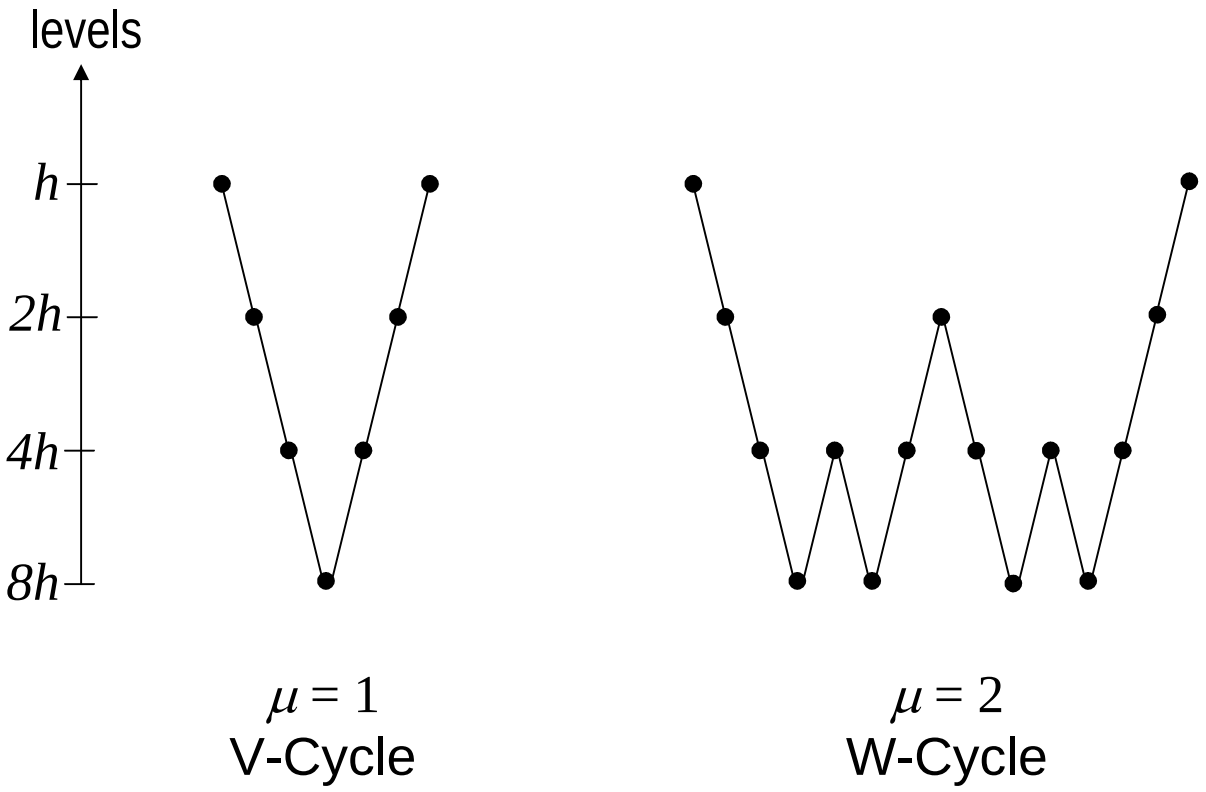
→ Idea: Recursively go down/ up more levels

$\Omega^h \rightarrow \Omega^{2h} \rightarrow \Omega^{4h} \rightarrow \ldots \rightarrow \Omega^{ph} \rightarrow \Omega^{0.5ph} \rightarrow \ldots \rightarrow \Omega^h$

μ-Cycle scheme (multigrid)

$$v^h \leftarrow M\mu\left(v^h, f^h\right)$$

1. Relax ν_1 times on $A^h u^h = f^h$ with a given initial guess v^h (*pre-smoothing*)
2. If Ω^h = coarsest grid, then go to 4.
   Else $f^{2h} \leftarrow I_h^{2h}\left(f^h - A^h v^h\right)$
   $v^{2h} \leftarrow 0$
   $v^{2h} \leftarrow M\mu\left(v^{2h}, f^{2h}\right)$ μ times
3. Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$
4. Relax ν_2 times on $A^h u^h = f^h$ with initial guess v^h (*post-smoothing*)



But: Regard V-Cycle:
Problem: What vector to choose as initial guess?

→ Idea: Combine V-Cycle and nested iteration

Reminder: Nested iteration
Main idea: Use a coarse grid to obtain a good initial guess for the "original" grid

→ Iterate following scheme:
- Find solution on coarsest grid
- Go to the next finer grid and do one V-cycle

Full multigrid V-Cycle

Let $v^h \leftarrow MV^h\left(v^h, f^h\right)$ be the V-Cycle scheme.

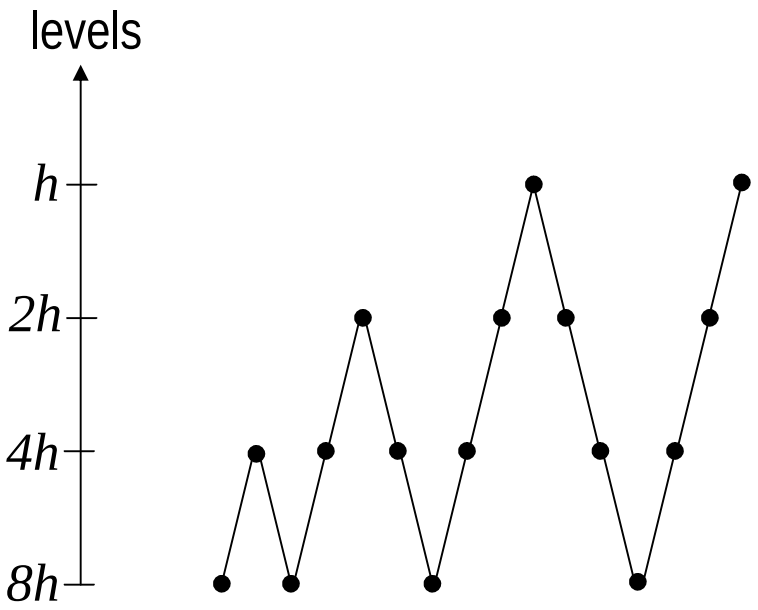
$$v^h \leftarrow FMV^h\left(v^h, f^h\right)$$

1. If Ω^h = coarsest grid, then go to step 3.

   Else $f^{2h} \leftarrow I_h^{2h}\left(f^h - A^h v^h\right)$
   $v^{2h} \leftarrow 0$
   $v^{2h} \leftarrow FMV^{2h}\left(v^{2h}, f^{2h}\right)$.

2. Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$.

3. $v^h \leftarrow MV^h\left(v^h, f^h\right)$ ν_0 times.



Full multigrid V-Cycle

Performance of multigrid schemes

How to implement multigrid

- Relaxation, interpolation, restriction can be implemented in individual modules
- As manageable data structure: choose single arrays (for regular grids)

  ⟶ Let $\Omega := \left\{ (ih, jh), i, j = 1,...n, h := \frac{1}{n-1} \right\}$ be a grid of the domain $(0,1)^2$

  ⟶ create an array $u[\,]$ of length n^2
  point $u(ih, jh) \mathrel{\hat{=}} u[i \cdot n + j]$

- For each grid: 2 arrays for approximate solution (incl. boundary values) and right-hand side

Initialisations (V-Cycle)

- set $v^h = \vec{0},\ f^{kh} = \vec{0}\ \ (k > 1)$

- V-Cycle "descends" ⟶ solution $v^{kh} = \vec{0},\ \ (k > 1)$ is filled by relaxation scheme
  ⟶ residual transfer fills right-hand side

- V-Cycle "ascends" ⟶ right-hand side is not changed
  ⟶ solution is corrected

Complexity (storage)

- storage cost = $2N^d\left(1+2^{-d}+2^{-2d}+\ldots+2^{-pd}\right)<\frac{2N^d}{1-2^{-d}}$

Complexity (computational costs)

- introduce work units WU (= cost of performing 1 relaxation sweep on finest grid)
- neglect cost of intergrid operations (ca. 15% - 20% of entire costs)
- V-Cycle ($\nu_1=\nu_2=1$):

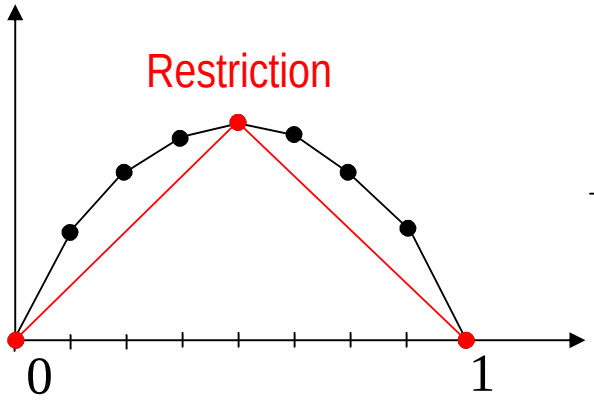
  $\longrightarrow$ comp. cost = $2\left(1+2^{-d}+2^{-2d}+\ldots+2^{-pd}\right)\text{WU}<\frac{2}{1-2^{-d}}\ \text{WU}$
- FMV-Cycle:

  $\longrightarrow$ comp. cost = $\frac{2}{1-2^{-d}}\left(1+2^{-d}+2^{-2d}+\ldots+2^{-pd}\right)\text{WU}<\frac{2}{\left(1-2^{-d}\right)^2}\text{WU}$

Convergence of multigrid

- Main advantage of multigrid: Regard again $-f'' = 0$ on (0,1)

$$f(0) = f(1) = 0$$



→ Turn a low frequency curve on a fine grid into a high frequency curve on a coarser grid

→ Multigrid improves convergence by improving the behaviour of the relaxation schemes

- In the following: Find conditions for the error $e := \|u - v^h\|$ and the meshsize h

- Assume, that e shall be smaller than some tolerance ε

  → Idea: Split the error in two parts and require that each part shall be smaller than $\varepsilon/2$:

$$e = \|u - v^h\| \leq \underbrace{\|u - u^h\|}_{\text{discretization error}} + \underbrace{\|u^h - v^h\|}_{\text{algebraic error}} < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon$$

- From discretization process, we achieve for the discretization error:

$$E^h := \|u - u^h\| \leq K \cdot h^p, \quad K \in \mathrm{IR} \quad \Rightarrow \quad h < h^* = \left(\frac{\varepsilon}{2K}\right)^{\frac{1}{p}}$$

- Question: How many cycles have to be performed to get the error smaller than the set tolerance?

- Goal: Reduce algebraic error of $O(1)$ ($\rightarrow$ initial guess) to $O(h^p) = O(N^{-p})$

- Consider a V-Cycle with convergence rate γ (independent from h!), and let υ be the needed number of iterations. Then, we get

$$\gamma^{\upsilon} = O\left(N^{-p}\right) \Rightarrow \upsilon = O(\log N).$$

- Since the cost of a single V-Cycle is $O(N^d)$, it follows:

> Convergence to the level of truncation is achieved by the V-Cycle in $O(N^d \log N)$.

- Consider FMV: before Ω^h-problem is regarded, Ω^{2h}-problem is solved to the level of truncation

$\longrightarrow$ Number of cycles is $\upsilon = O(1)$, FMV-cost is $O(N^d)$

Remarks on V-Cycles and FMV:
- One single FMV costs more than a V-cycle, but it is also probably more effective
- In general: FMV more preferable

Literature:
- W.L. Briggs, A Multigrid Tutorial, SIAM, Philadelphia, 1987
- C. Pflaum, Simulation And Scientific Computing II, Erlangen, 2006

Code example 2

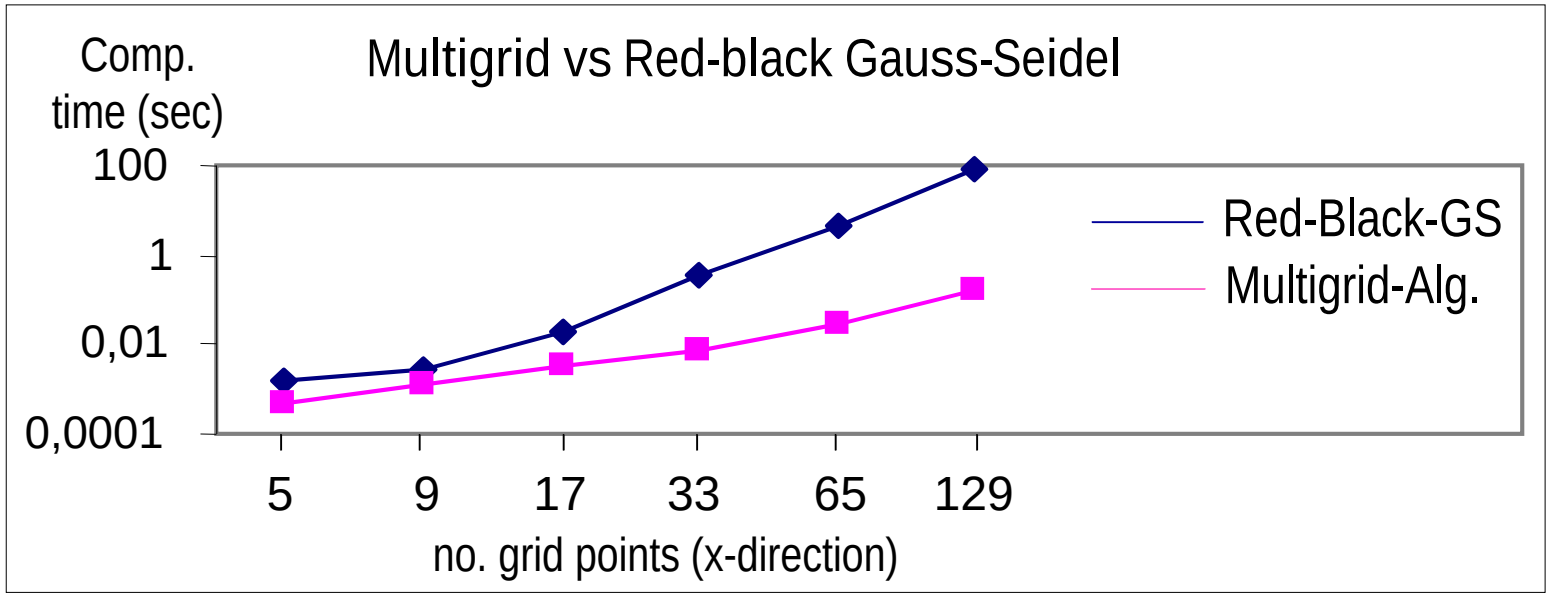
Task (taken from the Assignments of "Simulation and scientific computing II"):

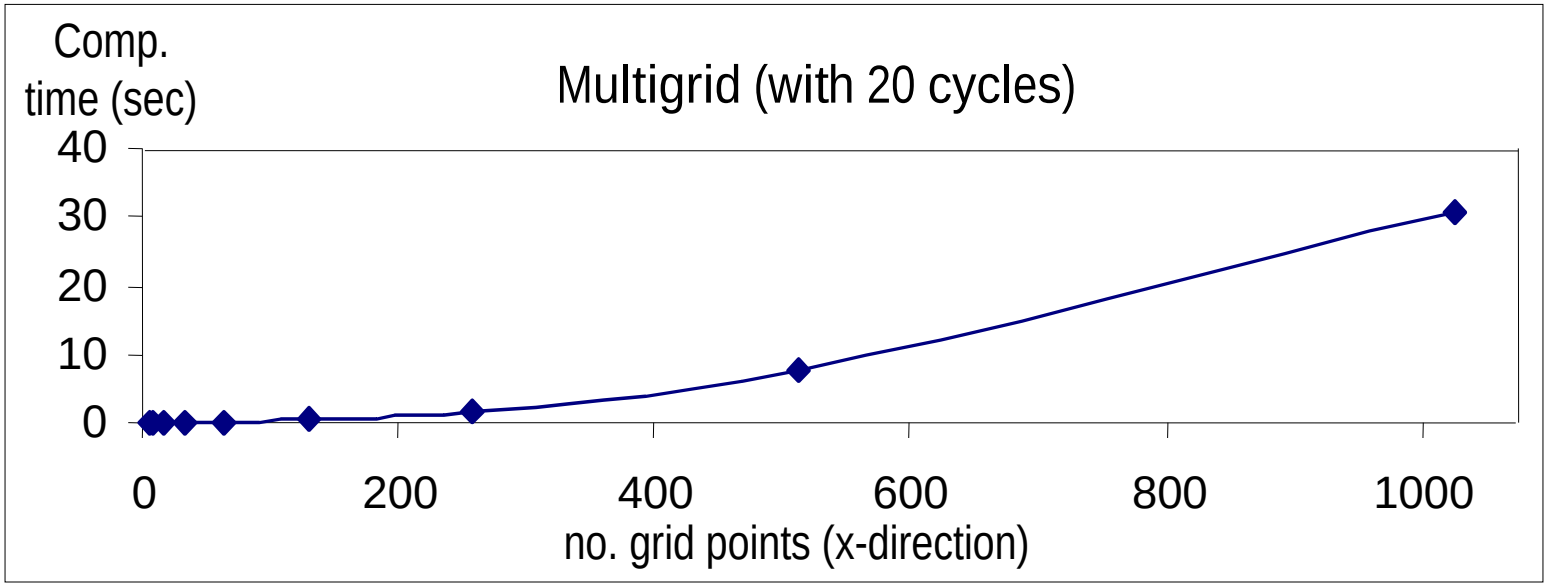
Write a program "*mgsolve.cpp*" that discretizes the PDE

$$-\Delta u = 32\pi^2 \cos(4\pi x)\cos(4\pi y) \quad \text{on } \Omega := (0,1)^2$$
$$u = \cos(4\pi x)\cos(4\pi y) \quad \text{on } \partial\Omega$$

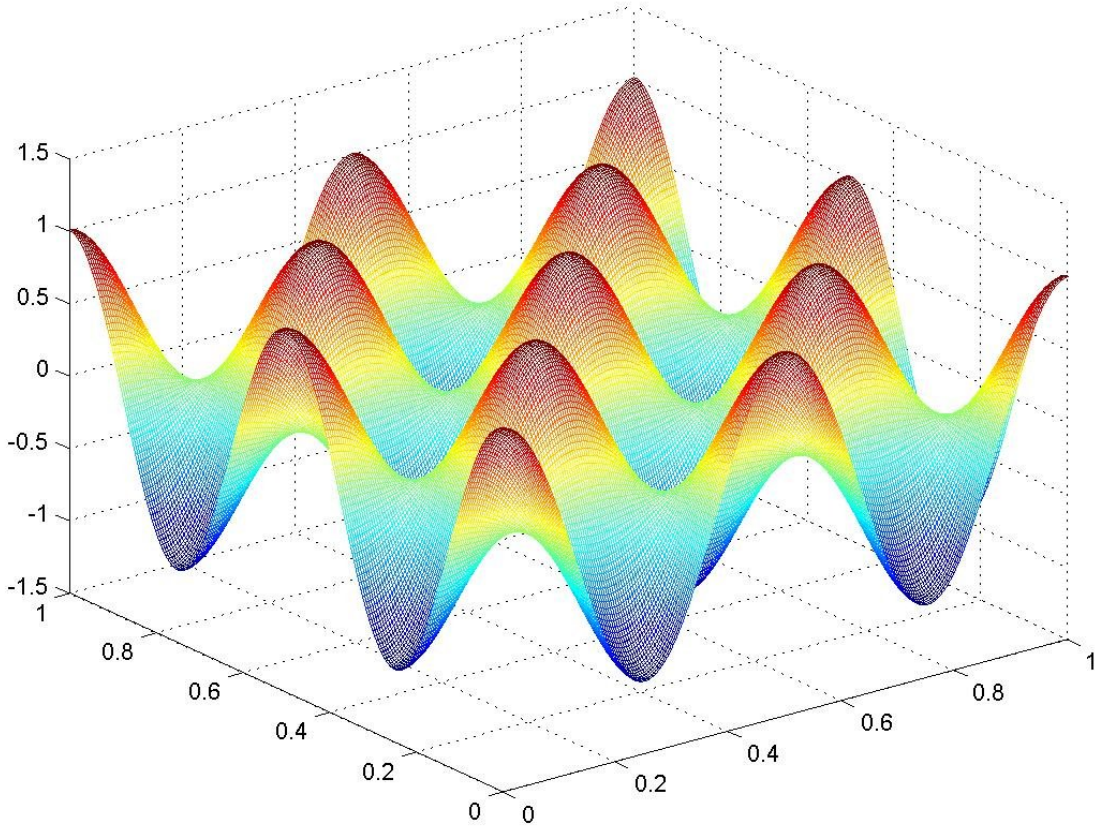
by finite differences on $\overline{\Omega}$ with 2^l+1 **x** 2^l+1 grid points and solves the discrete problem by V(2,1)-cycles, using l levels and $u = 0$ as initial guess inside the domain. Use full weighting for restriction, (bilinear) interpolation for prolongation and red-black Gauss-Seidel for relaxation. The programm should finally create the file *"mgsolution.txt"* and write there the approximate solution in order to visualize it later.

→ Let's have a look at the code!





And the main thing is...



... you get nice coloured pictures!